Knowledge representation

lecture 2:

How is SQL related to logic

T. Tammet

Side remarks:

declarative and procedural knowledge

Declarative and procedural

In psychology:

Declarative knowledge is knowing „what" (e.g., that Washington D.C. is the capital of America),

Procedural knowledge is knowing "how" (e.g., how to drive a car).

Declarative and procedural

In programming:

Declarative representation: data and rules written in an easily processable simple format.

Procedural representation: data and rules written as an executable program.

A strict distinction is - obviously - impossible

Layers of declarative data

First layer: **plain facts** like

company_id: Mycompany

limit1: 64

Second layer: **rules** like

if (X1<123 || X2<65) then result=12

Layers in psychology

Episodic knowledge: memory for "episodes" (i.e., the context of where, when, who with etc); usually measured by accuracy measures, has autobiographical reference.

Semantic knowledge: Memory for knowledge of the world, facts, meaning of words, etc. (e.g., knowing that the first month of the year is April (alphabetically) but January (chronologically).

No strict demarcation line

Attempting to make everything “configurable” leads us to using just another programming language for expressing configurable data and rules!

Ordinary programming languages are also seen as “declarative”, at least by compilers.

Transformation in our brains

Some initially **declarative** knowledge converted / optimised to **procedural**

Learning to drive a car, play tennis, ...

“For example, when I was learning to play tennis, I learned all about the rules of the game, where to come into contact with the ball on my racket, how to make the ball go where I wanted to by the follow through, and how to position my body for a backhand stroke. This is a set of factual information. Putting those facts into practice helped me gain the skills to transform a series of declarative knowledge into procedural knowledge. The skills I acquired couldn't be learned simply by being told. I gained the skills only after actively putting them into practice and being monitored by a coach who was constantly providing feedback. “

Transformation in programming

Declarative knowledge converted to procedural

for easier automation and more speed

- Rough understanding, notes and spec of the solution written down as an actual program code.
- Source code compiled into machine code.

Viewing relational databases as logic

Presentation plan

- refresher of 1st order predicate logic
- meaning of data in databases
- tables as predicates: straightforward encoding
- queries and joins as rules
- special objects and features: multiple rows, null
- encoding structures in db-s
- db keys as a way to encode functions

Pred calculus example 1: prolog

Data:

```
father(jan,pete).

father(jan,martin).

father(martin,matt).

father(frank,mary).

mother(mary,mike).
```

Rules:

```
grandfather(X,Z) :- father(X,Y), father(Y,Z).

grandfather(X,Z) :- father(X,Y), mother(Y,Z).
```

Queries:

```
? father(martin,mike)

? father(jan,X)

? grandfather(X,mike)
```

Pred calculus example 2

brother(jim,pete).

Forall X,Y (brother(X,Y) => brother(Y,X)). (?? check !)

Forall X,Y (brother(X,Y) => man(X)).

query: man(X)

answers: man(jim), man(pete)

Pred calculus with functions

Data:

```
father(pete)=jan.

father(martin)=jan.

father(matt)=martin.

father(mary)=frank.

mother(mike)=mary.
```

Rules:

```
grandfather(X,Z) <- father(Y)=X & father(Z)=Y

alternative: grandfather(father(father(X)),X).

grandfather(X,Z) <- father(Y)=X & mother(Z)=Y

alternative: grandfather(father(mother(X)),X).
```

Words in logic have no meaning

Data:

```
foo(p1)=j.

foo(m1)=j).

foo(m2)=m1.

foo(m3)=f.

bar(m4)=m3.
```

Rules:

```
grm(X,Z) <- foo(Y)=X & foo(Z)=Y

alternative: grm(foo(foo(X)),X).

grm(X,Z) <- foo(Y)=X & bar(Z)=Y

alternative: grm(foo(bar(X)),X).
```

Meaning of words?

Relations give meaning to words

What about = ?? father(john)=pete then just replace .

Three basic rules:

e(X,X)

e(X,Y) -> e(Y,X)

e(X,Y) & e(Y,Z) -> e(X,Z)

Examples: parallel lines, >=, relative

Meaning of words?

Substitution rules of equality:

e(X,Y) & father(X,Z) -> father(Y,Z)

e(X,Y) & father(Z,X) -> father(Z,Y)

e(X,Y) & e(father(X),Z) -> e(father(Y),Z)

Built-in procedures and theories for relations and functions

- Arithmetics + * etc: procedural attachments (procedural data representation)
- String, list, date, file etc etc proc attachments
- Special built-in theories like Presburger arithmetics

Solvers focusing on procedural attachments and built-in theories are called **SMT** solvers: **solvers modulo theories**

Relational db table and logic

Client table:

id	name	balance
1	john	100
2	pete	-200

Logic:

client(1, john, 100)

client(2, pete, -200)

Queries

```
select
client.name, client.balance
from client
where  balance<0;
```

```
client(I, N, B)  & B<0   ->  answer(N, B)
```

```
? answer(X, Y)
```

SQL join as logical rule

client table:

id	name	balance
1	john	100
2	pete	-200

cars table:

id	model	owner
1	ford	1
2	opel	2
3	saab	2

```
select client.name, cars.model from client, cars
where client.id=cars.owner
```

client(I, N, B) & cars(J, M, I) -> ans(N, M)

Representing complex structures in relational databases

+(*(1.9, 2.5),3)

Term:

id	op	a1	t1	a2	t2
1	*	2	D	1	D
2	+	1	T	3	I

Data:

	varchar	type
1	“2.5”	F
2	“1.9”	F
3	“3”	I

Special relational db gadgets

These things require extra care when encoding in logic:

- null value
- keys
- multiple rows
- closed world

null values

null values in two different locations are never equal!

client(1, jaan, null) and car(1, opel, null) then null in client is not equal to null in car

null represents an existentially quantified var:

Exists X. client(1, jaan, X).

Exists X. car(1, opel, X).

Keys

Client table:

id	name	balance
1	jan	100
2	pete	-200

ALTER TABLE client ADD PRIMARY KEY (id)

Would mean in logic:

Client.name(1) = jan

Client.name(2) = pete

Client.balance(1) = 100

….

Multiple rows

payments table without a primary key

client	sum
1	100
1	100
2	50

two identical facts in logic mean a single fact:

payments(1, 100) …. payments(1, 100)

Open versus closed world

Classical logic is **open**:

- We know N facts. We do NOT say that only these N facts hold: maybe there are M more facts which are true but which we do not know.

In databases we normally assume that world is **closed**:

• Only these facts hold which we know. For example, we assume that all the payments performed are in our database. This allows us to use **aggregate functions** like sum, avg, ...